Self-Healing of Operational Workflow Incidents on Distributed Computing Infrastructures

Rafael Ferreira da Silva, Tristan Glatard, Frédéric Desprez

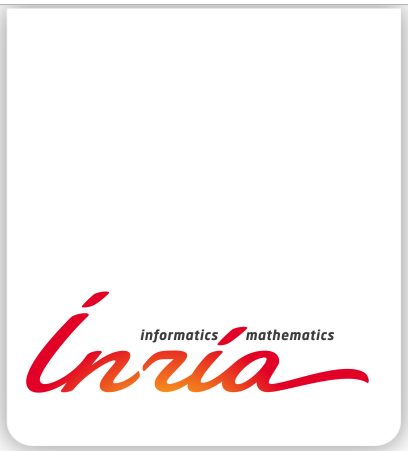
Self-Healing of Operational Workflow Incidents on Distributed Computing Infrastructures

Rafael Ferreira da Silva[1], Tristan Glatard[1], Frédéric Desprez[2]

[1]CNRS, University of Lyon, CREATIS, Villeurbanne, France
[2]INRIA, University of Lyon, LIP, ENS Lyon, Lyon, France

RESEARCH REPORT
N° 8022
July 2012
Project-Team Avalon

ISSN 0249-6399 ISRN INRIA/RR--8022--FR+ENG



Self-Healing of Operational Workflow Incidents on Distributed Computing Infrastructures

Rafael Ferreira da Silva*[1], Tristan Glatard†[1], Frédéric Desprez‡[2]

[1]CNRS, University of Lyon, CREATIS, Villeurbanne, France
[2]INRIA, University of Lyon, LIP, ENS Lyon, Lyon, France

Project-Team Avalon

Research Report n° 8022 — July 2012 — 27 pages

Abstract: Distributed computing infrastructures are commonly used through scientific gateways, but operating these gateways requires important human intervention to handle operational incidents. This report presents a self-healing process that quantifies incident degrees of workflow activities from metrics measuring long-tail effect, application efficiency, data transfer issues, and site-specific problems. These metrics are simple enough to be computed online and they make little assumptions on the application or resource characteristics. From their degree, incidents are classified in levels and associated to sets of healing actions that are selected based on association rules modeling correlations between incident levels. We specifically study the long-tail effect issue, and propose a new algorithm to control task replication. The healing process is parametrized on real application traces acquired in production on the European Grid Infrastructure. Experimental results obtained in the Virtual Imaging Platform show that the proposed method speeds up execution up to a factor of 4, consumes up to 26% less resource time than a control execution and properly detects unrecoverable errors.

Key-words: Error detection and handling, workflow execution, production distributed systems

* rafael.silva@creatis.insa-lyon.fr
† glatard@creatis.insa-lyon.fr
‡ Frederic.Desprez@inria.fr

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Auto-réparation d'incidents opérationnels lors de l'exécution de workflows sur des infrastructures distribuées

Résumé : Les infrastructures de calcul distribué sont couramment utilisées à travers des environnements applicatifs dédiés, mais l'administration de ces environnements demande un effort humain important pour résoudre les incidents qui surviennent en production. Ce rapport présente une méthode d'administration automatique qui quantifie le degré des incidents touchant les activités des chaînes de traitements. Ce degré est obtenu à partir de métriques mesurant le retard des dernières tâches, l'efficacité de l'application, les problèmes de transfert de données et la spécificité d'un incident à un site. Ces métriques sont suffisamment simples pour être calculées en ligne, et elles font très peu d'hypothèses sur les caractéristiques des applications et des ressources. A partir de leur degré, les incidents sont classés en niveaux et associés à des ensembles d'actions sélectionnées à partir de règles d'association qui modélisent la corrélation entre niveaux. Nous étudions particulièrement le retard des dernières tâches et nous proposons un algorithme pour contrôler leur réplication. Notre méthode d'administration automatique est paramétrée à partir de traces d'applications réelles acquises en production sur l'infrastructure de grille européenne (EGI). Des résultats expérimentaux obtenus sur la Plate-forme d'Imagerie Virtuelle (VIP) montrent que la méthode peut accélérer l'exécution jusqu'à un facteur 4, économise 26% de ressources par rapport à une exécution-témoin, et détecte correctement les incidents qui ne peuvent pas être résolus.

Mots-clés : Détection et résolution d'erreur, exécution de chaînes de traitements, systèmes distribués en production

1 Introduction

Distributed computing infrastructures (DCI) are becoming daily instruments of scientific research, in particular through scientific gateways [18] developed to allow scientists to transparently run their analyses on large sets of computing resources. While these platforms provide important amounts of resources in an almost seamless way, their large scale and the number of middleware systems involved lead to many errors and faults. Easy-to-use interfaces provided by these gateways exacerbate the need for properly solving operational incidents encountered on DCIs since end users expect high reliability and performance with no extra monitoring or parametrization from their side. In practice, such services are often backed by substantial support staff who monitors running experiments by performing simple yet crucial actions such as rescheduling tasks, restarting services, killing misbehaving runs or replicating data files to reliable storage facilities. Fair QoS can then be delivered, yet with important human intervention.

For instance, the long-tail effect [8] is a common frustration for users who have to wait for a long time to retrieve the last few pieces of their computations. Operators may be able to address it by rescheduling tasks that are considered late (e.g. due to execution on a slow machine, low network throughput or just loss of contact) but detection is very time consuming and still approximate.

Automating such operations is challenging for two reasons. First, the problem is online by nature because no reliable user activity prediction can be assumed, and new workloads may arrive at any time. Therefore the considered metrics, decisions and actions have to remain simple and to yield results while the application is still executing. Second, it is non-clairvoyant due to the lack of information about applications and resources in production conditions. Computing resources are usually dynamically provisioned from heterogeneous clusters, clouds or desktop grids without any reliable estimate of their availability and characteristics. Models of application execution times are hardly available either, in particular on heterogeneous computing resources.

A scientific gateway is considered here as a platform where users can process their own data with predefined applications workflows. Workflows are compositions of *activities* defined independently from the processed data and that only consist of a program description. At runtime, activities receive data and spawn invocations from their input parameter sets. Invocations are assumed independent from each other (bag of tasks) and executed on the DCI as single-core tasks which can be resubmitted in case of failures. This model fits several existing gateways such as e-bioinfra [31], P-Grade [23], and the Virtual Imaging Platform [14]. We also consider that files involved in workflow executions are accessed through a single file catalog but that storage is distributed. Files may be replicated to improve availability and reduce load on servers.

The gateway may take decisions on file replication, resource provisioning, and task scheduling on behalf of the user. Performance optimization is a target but the main point is to ensure that correctly-defined executions complete, that performance is acceptable, and that misbehaving runs (e.g. failures coming from user errors or unrecoverable infrastructure downtimes) are quickly detected and stopped before they consume too many resources.

Our ultimate goal is to reach a general model of such a scientific gateway that could autonomously detect and handle operational incidents. In this work,

we propose a healing process for workflow activities only. Activities are modeled as Fuzzy Finite State Machines (FuSM) [27] where state degrees of membership are determined by an external healing process. Degrees of membership are computed from metrics assuming that incidents have outlier performance, e.g. a site or a particular invocation behaves differently than the others. Based on incident degrees, the healing process determines incident levels using thresholds determined from platform history. A specific set of actions is then selected from association rules among incident levels. We specifically study the long-tail effect issue, and propose a new algorithm to control task replication.

Section 2 presents related work. Our approach is described in section 3 (general healing process), section 4 (metrics used to quantify incident degrees) and section 5 (incident levels and associated action sets). Experimental results are presented in section 6 in production conditions.

2 Related Work

2.1 Autonomic Computing

Managing systems with limited intervention of system administrators is the goal of autonomic computing [24]. It has been used to address various problems related to self-healing, self-configuration, self-optimization, and self-protection of distributed systems. For instance, provisioning of virtual machines is studied by Nguyen et al. [28] and an approach to tackle service overload, queue starvation, "black hole" effect and job failures is sketched by Collet et al. [9].

An autonomic manager consists of monitoring, analysis, planning, execution and knowledge (so-called MAPE-K loop). Generic software frameworks have been built to wrap legacy applications in such loops with limited intrusiveness. For instance, Broto et al. [4] demonstrates the wrapping of DIET grid services for autonomic deployment and configuration. We consider here that the target gateway can be instrumented to report appropriate events and perform actions.

Monitoring is broadly studied in distributed systems, both at coarse (traces, archives) and fine time scales (active monitoring, probing). Many workload archives are available. In particular, the grid observatory [17] has been collecting traces for a few years on several grids. However, as noted by Iosup & Epema [22], most existing traces remain at the task level and lack information about workflows and activities. Application patterns can be retrieved from logs (e.g. bag of tasks) but precise information about workflow activities is bound to be missing. Studies on task errors and their distributions are also available [26, 25], but they do not consider operational issues encountered by the gateways submitting these tasks. Besides, active monitoring using tools such as Nagios [21] cannot be the only monitoring source when substantial workloads are involved. Therefore we rely on traces of the target gateway, as detailed in section 5. One issue in this case is to determine the timespan where system behavior can be considered steady-state. Although this issue was recently investigated [13], it remains difficult to identify non-stationarities in an online process and we adopt here a stationary model.

Analysis consists in computing metrics (a.k.a. utility functions) from monitoring data to characterize the state of the system. System state usually distinguishes two regimes: properly functioning and misfunctioning. Zhang et al. [34]

assume that incidents lead to non stationarity of the workload statistics and use the Page-Hinkely test to detect them. Stehle et al. [32] present a method where the convex hull is used instead of hyper-rectangles to classify system states. As described in section 5, we use multiple threshold values for a given metric to use more than two levels to characterize incidents.

Planning and actions considered in this work deal with task scheduling and file replication. In these domains, most approaches are clairvoyant, meaning that resource, task, error rate and workload characteristics are precisely known [3, 20]. Heuristics are designed by Casanova et al. [7] for the case where only data transfer costs are known, on an offline problem though. Quintin and Wagner [29] propose an online task scheduling algorithm where some characteristics of the application DAG is known in advance. Camarasu-Pop et al. [5] propose a dynamic load-balancing strategy proposed to remove the long-tail effect on production heterogeneous systems, but it is limited to Monte-Carlo simulations.

The general task scheduling problem is out of our scope. We assume that a scheduler is already in place, and we only aim at performing actions when it does not deliver proper performance. In particular, we focus on site blacklisting and on task dynamic replication [16] to avoid long-tail effect.

2.2 Task and File Replication

Task replication, a.k.a. redundant requests is commonly used to address non-clairvoyant problems [8], but it should be used sparingly to avoid overloading the middleware and degrading fairness among users [6]. In this work, task replication is considered only when activities are detected blocked or of low efficiency according to the metric presented in section 4.

An important, and not so considered, aspect to be evaluated is the resource waste, a.k.a. the cost of task replication. Cirne et al. [8] evaluates the waste of resources by measuring the percentage of wasted cycles among all the cycles required to execute the application.

File replication strategies also often assume clairvoyance on the size of produced data, file access pattern and infrastructure parameters [2, 12]. In practice, production systems mostly remain limited to manual replication strategies [30].

3 General Healing Process

An activity is modeled as an FuSM with 13 states shown on Figure 1. The activity is initialized in `Submitting Invocations` where all the tasks are generated and submitted. Tasks consist of 4 successive phases: initialization, inputs download, application execution and output upload. They are all assumed independent, but with similar execution times (bag of tasks). `Running` is a state where no particular issue is detected; no action is taken and the activity is assumed to behave normally. `Completed` (resp. `Failed`) is a terminal state used when all the invocations are successfully completed (resp. at least one invocation failed). These 4 states are crisp (not fuzzy) and exclusive. Their degree can only be 0 or 1 and if 1 then all the other states have a degree of 0. The 9 other states are fuzzy states corresponding to detected incidents.

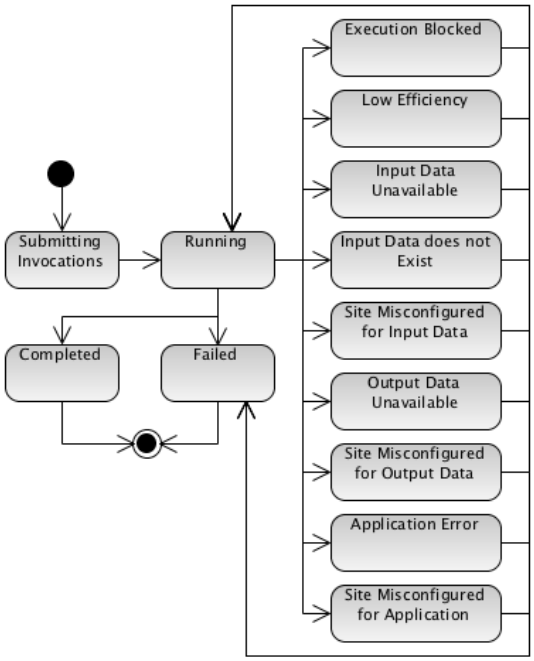


Figure 1: Fuzzy Finite State Machine (FuSM) representing an activity.

The healing process sets the degree of FuSM states from incident detection metrics and invocation statuses. Then it determines the actions to be performed to address the incidents. If no action is required then the process waits until an event occurs (task status change) or a timeout is reached.

Let $I = \{x_i, i = 1, \ldots, n\}$ be the set of possible incidents (9 in this work) and $\eta = (\eta_1, \ldots, \eta_n) \in [0, 1]^n$ their degrees in the FuSM. Incident x_i can occur at m_i different levels $\{x_{i,j}, j = 1, \ldots, m_i\}$ delimited by threshold values $\tau_i = \{\tau_{i,j}, j = 1, \ldots, m_i\}$. The level of incident i is determined by j such that $\tau_{i,j} \leq \eta_i < \tau_{i,j+1}$. A set of actions $a_i(j)$ is available to address $x_{i,j}$:

$$\begin{aligned} a_i : [1, m_i] &\rightarrow \wp(A) \\ j &\mapsto a_i(j) \end{aligned} \tag{1}$$

where A is the set of possible actions taken by the healing process and $\wp(A)$ is the power set of A.

In addition to the incidents themselves, incident causes are taken into account. Association rules [1] are used to identify relations between levels of different incidents. Association rules to $x_{i,j}$ are defined as $R_{i,j} = \{r_{i,j}^{u,v} = (x_{u,v}, x_{i,j}, \rho_{i,j}^{u,v})\}$. Rule $r_{i,j}^{u,v}$ means that when $x_{u,v}$ happens then $x_{i,j}$ also happens with confidence $\rho_{i,j}^{u,v} \in [0, 1]$. The confidence of a rule is an estimate of probability $P(x_{i,j}|x_{u,v})$. Note that $r_{i,j}^{i,j} \in R_{i,j}$ and $\rho_{i,j}^{i,j} = 1$. We also define $R = \bigcup_{i \in [\![1,n]\!], j \in [\![1,m_i]\!]} R_{i,j}$.

Figure 2 presents the algorithm used at each iteration of the healing process. Incident degrees are determined based on metrics presented in section 4 and incident levels j are obtained from historical data as explained in section 5. A roulette wheel selection [11] based on η is then performed to select $x_{i,j}$ the incident level of interest at this iteration. In a roulette wheel selection, incident x_i

is selected with a probability p_i proportional to its degree: $p(x_i) = \eta_i / \sum_{j=1}^{n} \eta_j$. A potential cause $x_{u,v}$ for incident $x_{i,j}$ is then selected from another roulette wheel selection on the association rules $r_{i,j}^{u,v}$, where x_u is at level v. Rule $r_{i,j}^{u,v}$ is weighted $\eta_u \times \rho_{i,j}^{u,v}$ in the roulette selection. Only first-order causes are considered here but the approach could be extended to include more recursion levels. Note that $r_{i,j}^{i,j}$ participates in this selection so that a first-order cause is not systematically chosen. Finally, actions in $a_u(v)$ are performed.

Input: invocation statuses and history of η
Output: set of actions a

01. wait for event or timeout
02. determine incident degrees η based on metrics
03. determine incident levels j such that $\tau_{i,j} \leq \eta_i < \tau_{i,j+1}$
04. select incident x_i by roulette wheel selection based on η
05. select rule $r_{u,v} = (x_{u,v}, x_{i,j}, \rho_{i,j}^{u,v}) \in R_{i,j}$ by roulette wheel selection based on $\eta_u \times \rho_{i,j}^{u,v}$, where x_u is at level v
06. $a = a_u(v)$
07. perform actions in a

Figure 2: One iteration of the healing process.

Table 1 illustrates this mechanism on an example case where only 3 incidents are considered, and Figure 3 shows it as a MAPE-K loop.

Step 02 and 03: incident degrees and levels are determined:

x_i: incident name	Degree η_i	Level j
x_1: activity blocked	0.8	2
x_2: low efficiency	0.4	1
x_3: input data unavailable	0.1	1

Step 04: $x_{1,2}$ is selected with probability $\frac{0.8}{0.8+0.4+0.1}$.

Step 05: association rules $r_{1,2}^{2,1}$, $r_{1,2}^{3,1}$ and $r_{1,2}^{1,2}$ are considered:

Rule	Confidence
$r_{1,2}^{2,1}$: $x_{2,1} \rightarrow x_{1,2}$	0.8
$r_{1,2}^{3,1}$: $x_{3,1} \rightarrow x_{1,2}$	0.2
$r_{1,2}^{1,2}$: $x_{1,2} \rightarrow x_{1,2}$	1

$r_{1,2}^{2,1}$ is chosen with probability $\frac{0.8 \times 0.4}{0.8 \times 0.4 + 0.2 \times 0.1 + 0.8 \times 1}$.

Step 06: actions in $a_2(1)$ are performed.

Table 1: Example case.

4 Incident Degree

This section describes the metrics used to determine the degree of the 9 considered incidents (step 02 on Figure 2).

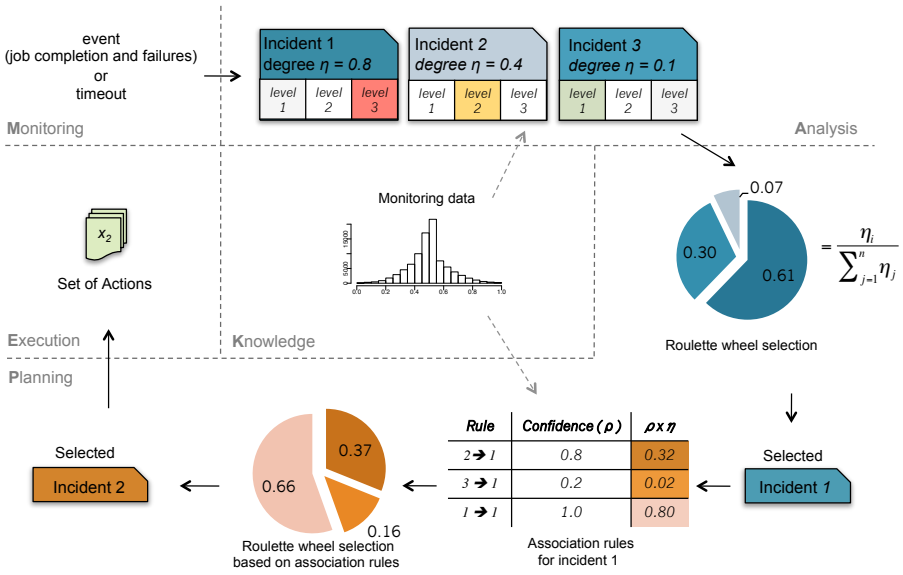


Figure 3: Example case showed as a MAPE-K loop.

4.1 Activity Blocked

This incident happens when an invocation is considered late compared to the others. It is responsible for many operational issues, leading to substantial speed-up reductions. For instance, it occurs when one invocation of the activity requires more CPU cycles or when the invocation faces longer waiting times, lost tasks or executes on resources with poorer performance. Two methods are proposed to cope with blocked activities. The first, called `Slope Contraction`, identifies blocked activities by detecting decreases of the derivative along time of the number of completed tasks, and the second, called `Estimation by Median`, identifies blocked activities as the ones whose tasks are performing worse than the median of already completed tasks. In both cases, invocations are assumed of identical duration.

Slope Contraction: this situation is detected online from the number $n(t)$ of completed invocations at time t (see Figure 4). At time t, we compute the slope $a(t)$ of the regression line of $\{(t_i, n(t_i)), t_i \leq t\}$. If the iteration is triggered by a timeout instead of an event (see step `01` on Figure 2), then $(t, n(t)+1)$ is added to the regression set. This is meant to ensure that long-running invocations can be handled before they complete. We then define the incident degree η_b from the contraction rate of the linear regression slope:

$$\eta_{\mathrm{b}} = 1 - \frac{a(t)}{a_{\max}(t)}$$

where $a_{\max}(t)$ is the maximal value of $a(t)$ in $[0, t]$. $t = 0$ is the time when the activity is started, i.e., all the invocations are initialized. Note that the maximum degree $\eta_b = 1$ is reached when the activity is completely blocked ($\lim_{t\to\infty} a(t) = 0$). On the other hand, $\eta_b = 0$ is reached when $a(t) = a_{max}(t)$.

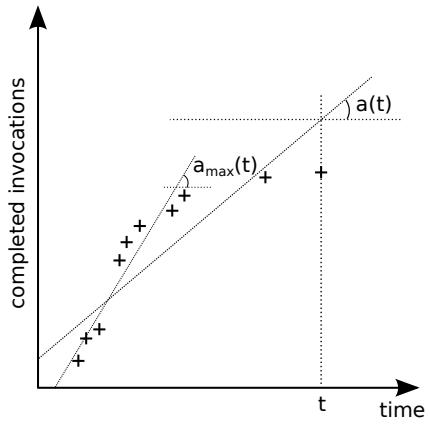


Figure 4: Detection of blocked activity.

Estimation by Median: we define the performance coefficient p_i of a task i relating the sum of the durations of the task phases (`setup`, `inputs download`, `application execution` and `outputs upload`) to the sum of the median durations of the task phases as follows:

$$\eta_{\mathrm{b}} = p_i = p(t_i, \tilde{t}) = \frac{t_i}{\tilde{t} + t_i}$$

where $t_i = t_{i\_setup} + t_{i\_input} + t_{i\_exec} + t_{i\_output}$ defines the estimated duration of task i and $\tilde{t} = \tilde{t}_{setup} + \tilde{t}_{input} + \tilde{t}_{exec} + \tilde{t}_{output}$ the sum of the median durations, computed on all the tasks of the activity. The incident degree η_b is defined by p_i. Note that $\lim p_{i\,t_i=+\infty} = 1$ and $p_i = 0$ when $t_i = 0$. For $p_i \leq 0.5$, the task is performing better or equivalent to the median. The task is considered blocked when p_i goes beyond a threshold.

The estimated duration (t_i) of a task is computed as follows: (i) if a phase is completed, then the actual consumed resource time is used; (ii) for ongoing phases the maximum value between the current consumed resource time and the median consumed time is taken; and (iii) for not started phases the time slot is filled by the median value. Figure 5 illustrates the estimation process of a task where the actual durations are used for the two first completed phases (42s for `setup` and 300s for `inputs download`), the `application execution` phase uses the maximum value between the current value of 20s and the median value of 400s, and the last phase (`outputs upload`) is filled by the median value of 15s, as it is not started yet.

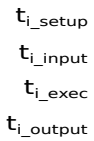


Figure 5: Task estimation based on median values.

4.2 Low Efficiency

This happens when the time spent by all the activity invocations in data transfers dominates CPU time. It may be due to sites with poor network connectivity or intrinsic to the application. The incident degree is defined from the ratio between the cumulative CPU time C_i consumed at time t by all completed invocations and the cumulative execution time at time t of all completed invocations:

$$\eta_{\text{e}} = 1 - \frac{\sum_{i=1}^{n(t)} C_i}{\sum_{i=1}^{n(t)} (C_i + D_i)}$$

where D_i is the time spent by invocation i in data transfers.

4.3 Input Data Unavailable

This happens when a file is registered in the file catalog but the storage resource(s) is(are) unavailable or unreachable. The incident degree η_{iu} in this state is determined from the input transfer failure rate due to data unavailability. Transfers of completed, failed, and running invocations are considered.

4.4 Input Data does not Exist

This happens when an incorrect data path was specified, the file was removed by mistake or the file catalog is unavailable or unreachable. Again, the incident degree η_{ie} is directly determined by the input transfer failure rate due to non-existent data. Transfers of completed, failed, and running invocations are considered.

4.5 Site Misconfigured for Input Data

This incident happens when sites have utmost input data transfer failure rate. The incident degree η_{is} at time t is measured as follows:

$$\eta_{\text{is}} = \max(\phi_1, \phi_2, \ldots, \phi_k) - \text{median}(\phi_1, \phi_2, \ldots, \phi_k)$$

where ϕ_i denotes the input transfer failure ratio (including both input data unavailable and input data does not exist) on site i at time t and k is the number of white-listed sites used by the activity at time t. The difference between the maximum rate and the median ensures that the incident degree has high values only when some sites are misconfigured. This metric is correlated but not redundant with the two previous ones. If some input data file is not available due to site-independent issues with the storage system, then η_{iu} will grow but η_{is} will remain low because all sites fail identically. On the contrary, η_{is} may grow while η_{iu} and η_{ie} remain low.

4.6 Output Data Unavailable

Output data can also be unavailable. Unavailability happens due to three main reasons: the user did not specify the output path correctly, the application did not produce the expected data, or the file catalog or storage resource are unavailable or unreachable. The incident degree η_{ou} is determined by the output

transfer failure rate. Transfers of completed, failed and running invocations are considered.

4.7 Site Misconfigured for Output Data

The incident degree η_{os} in this incident is determined as follows:

$$\eta_{\text{os}} = \max(\psi_1, \psi_2, \ldots, \psi_k) - \text{median}(\psi_1, \psi_2, \ldots, \psi_k)$$

where ψ_i denotes the output transfer failure ratio on site i at time t and k is the number of white-listed sites used by the activity at time t.

4.8 Application Error

Applications can fail due to a variety of reasons among which: the application executable is corrupted, dependencies are missing, or the executable is not compatible with the execution host. The incident degree η_{a} in this state is measured by the task failure rate due to application errors. Completed, failed, and running tasks are considered.

4.9 Site Misconfigured for Application

The incident degree η_{as} in this state is measured as follows:

$$\eta_{\text{as}} = \max(\alpha_1, \alpha_2, \ldots, \alpha_k) - \text{median}(\alpha_1, \alpha_2, \ldots, \alpha_k)$$

where α_i denotes the task failure rate due to application errors on site i and k is the number of white-listed sites used by the activity at time t.

5 Incident Levels and Actions

Incident degrees η_i are quantified in discrete incident levels so that different sets of actions can be used to address different levels of the incident. The threshold number and values are determined from observed distributions of η_i. The number m_i of incident levels associated to incident i is set as the number of modes in the distribution of η_i. Thresholds $\tau_{i,j}$ are determined from mode clustering.

5.1 Training Dataset

We used traces from the science-gateway workload archive [15] available to the community in the grid observatory[1]. Traces were collected from the Virtual Imaging Platform [14] between April and August 2011. Applications deployed in this platform are described as workflows executed using the MOTEUR workflow engine [19]. Resource provisioning and task scheduling is provided by DIRAC [33] using so-called "pilot jobs". Resources are provisioned online with no advance reservations. Tasks are executed on the biomed virtual organization (VO) of the European Grid Infrastructure (EGI)[2] which has access to some 150

[1] `http://www.grid-observatory.org`
[2] `http://www.egi.eu`

computing sites world-wide and to 120 storage sites providing approximately 4 PB of disk.

This data set contains 1,082 executions of 36 different workflows executed by 26 users. Workflow executions contain 1,838 activity instances, corresponding to 92,309 invocations and 123,025 tasks (including resubmissions).

Figure 6 shows the cumulative amount of running activities along this period. It shows that the workload is quite uniformly distributed although a slight increase is observed in June.

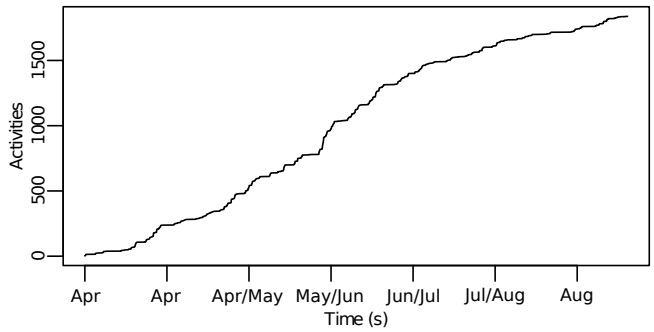


Figure 6: Cumulative amount of running activities from April to August 2011.

5.2 Incident Levels and Actions

Incident degrees were computed after each event found in this data set (total of 641,297 events). Figure 7 displays histograms of computed incident degrees. For readability purposes, only $\eta_i \neq 0$ values are represented. Histograms are clearly multi-modal, except for activity blocked identified by estimation, which confirms that incident degrees are quantified. Level numbers and threshold values τ are set from visual mode detection in these histograms and reported on Table 2 with associated actions. The threshold $\tau_{1,2}$ for task estimation was set to 0.66 to ensure that 90% of the running tasks are preserved.

Incidents at level 1 are considered painless for the execution and they do not trigger any action. Other levels can lead to radical (completely stop the activity or blacklist a site) or intermediate actions (task replication, file replication, or provisioning of extra resources).

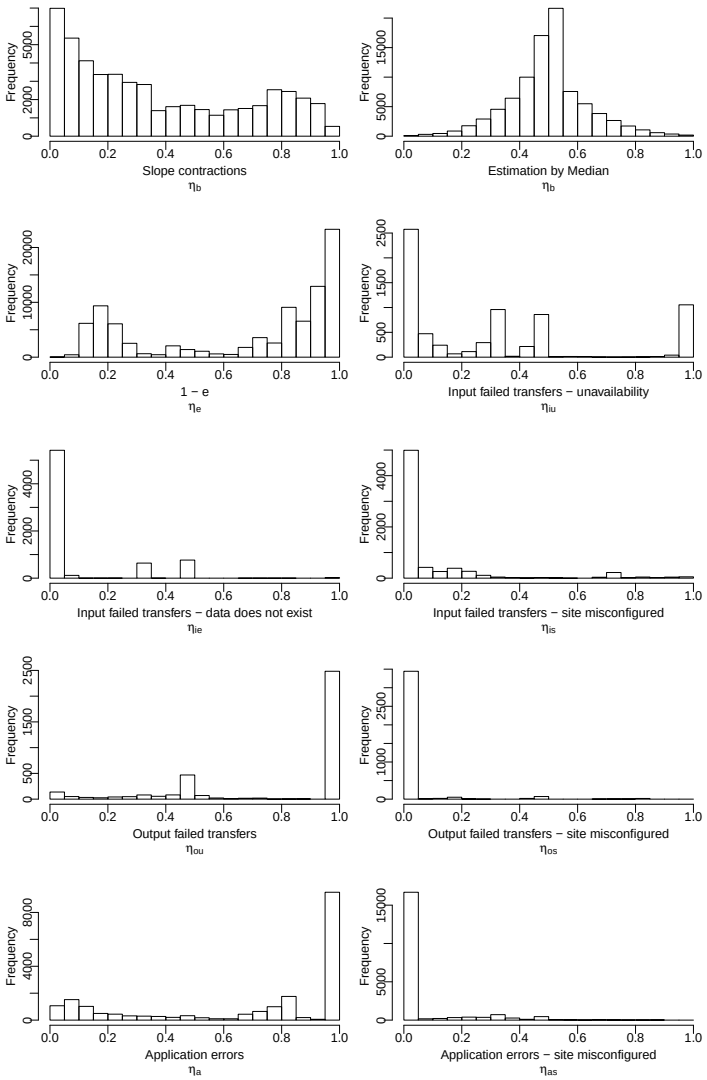


Figure 7: Histograms of incident degrees sampled in bins of 5%.

Incident (x_i)	Number of incident levels (m_i)	Level 1		Level 2		Level 3	
		$\tau_{i,1}$	actions	$\tau_{i,2}$	actions	$\tau_{i,3}$	actions
x_1: activity blocked (slope)	2	0	$\emptyset$	0.6	replicate running tasks		
x_1: activity blocked (estimation)	2	0	$\emptyset$	0.66	replicate running tasks		
x_2: low efficiency	2	0	$\emptyset$	0.6	replicate input files		
					replicate running tasks		
x_3: input data unavailable	3	0	$\emptyset$	0.2	replicate input files	0.8	stop activity
x_4: input data does not exist	2	0	$\emptyset$	0.8	stop activity		
x_5: site misconfigured for input data	3	0	$\emptyset$	0.3	replicate files on sites	0.65	blacklist site
					reachable from problematic site		
x_6: output data unavailable	2	0	$\emptyset$	0.8	stop activity		
x_7: site misconfigured for output data	2	0	$\emptyset$	0.1	blacklist site		
x_8: application error	2	0	$\emptyset$	0.5	stop activity		
x_9: site misconfigured for application	2	0	$\emptyset$	0.1	blacklist site		

Table 2: Incident levels and actions.

5.3 Task Replication

Triggering task replication can prevent slow nodes from delaying or halting the completion of tasks that can lead to activity blocked state. The `Slope Contraction` method has a greedy approach for task replication. However, an uncontrolled replication process can lead to resource waste. The replication process for a particular task in the `Estimation by Median` method is controlled by two mechanisms. First, a task is not replicated if a replica is already queued. Second, if replica j has better performance than replica r (i.e. $p(t_r, t_j) > \tau$) and j is a step further (i.e. is in a forward phase) than r, then replica r is aborted. Figure 8 presents the algorithm of the replication process for one task.

Input: Set of replicas R of a task i

01. rep = true
02. **for** $r \in R$ **do**
03. **for** $j \in R$, $j \neq r$ **do**
04. **if** $p(t_r, t_j) > \tau$ **and** j is a step further than r **then**
05. abort r
06. **done**
07. **if** r is started **and** $p(t_r, \tilde{t}) \leq \tau$ **then**
08. rep = false
09. **else if** r is queued **then**
10. rep = false
11. **done**
12. **if** rep == true **then**
13. replicate r

Figure 8: Replication process for one task.

5.4 Association Rules

Association rules are computed based on the frequency of occurrences of two incident levels. The confidence $\rho_{i,j}^{u,v}$ of a rule $x_{u,v} \Rightarrow x_{i,j}$ measures the probability that an incident level $x_{i,j}$ happens when $x_{u,v}$ occurs. Table 3 shows rule samples extracted from the training data-set and ordered by decreasing confidence. The set of rules leading to activity blocked ($x_{1,2}$) and low efficiency ($x_{2,2}$) incidents shows that they are partially dependent of other "cause" incidents, which is considered by the self-healing process.

At the bottom of the table we find rules with null confidence. These are consistent with common-sense interpretation of the incident dependencies (e.g. no site-specific issue when input data is unavailable).

6 Experiments

The healing process was implemented in the Virtual Imaging Platform (see description in section 5.1) and deployed in production. The experiments presented hereafter evaluate the ability of the healing process to (i) improve workflow

Association rule	$\rho_{i,j}^{u,v}$
$x_{5,2} \Rightarrow x_{2,2}$	0.3809
$x_{7,2} \Rightarrow x_{1,2}$	0.3529
$x_{5,3} \Rightarrow x_{1,2}$	0.3333
$x_{1,2} \Rightarrow x_{2,2}$	0.3059
$x_{3,2} \Rightarrow x_{1,2}$	0.2975
$x_{7,2} \Rightarrow x_{2,2}$	0.2941
$x_{5,2} \Rightarrow x_{1,2}$	0.2608
$x_{9,2} \Rightarrow x_{1,2}$	0.2435
$x_{2,2} \Rightarrow x_{1,2}$	0.2383
...	...
$x_{3,2} \Rightarrow x_{2,2}$	0.1276
$x_{7,2} \Rightarrow x_{3,3}$	0.1250
$x_{3,3} \Rightarrow x_{9,2}$	0.1228
$x_{7,2} \Rightarrow x_{3,2}$	0.0625
...	...
$x_{3,3} \Rightarrow x_{5,2}$	0.0000
$x_{3,3} \Rightarrow x_{5,3}$	0.0000
$x_{4,2} \Rightarrow x_{5,2}$	0.0000
$x_{4,2} \Rightarrow x_{5,3}$	0.0000
$x_{5,2} \Rightarrow x_{3,3}$	0.0000
$x_{5,2} \Rightarrow x_{4,2}$	0.0000
$x_{5,3} \Rightarrow x_{3,3}$	0.0000
$x_{5,3} \Rightarrow x_{4,2}$	0.0000

Table 3: Confidence of rules between incident levels.

makespan in case of recoverable incidents and (ii) quickly identify and report critical issues.

6.1 Implementation

The FuSM and healing process were implemented in the MOTEUR workflow engine. The timeout value in the healing process was computed dynamically as the median of the task inter-completion delays in the current execution.

Task replication is performed by resubmitting running tasks to DIRAC. To avoid concurrency issues in the writing of output files, a simple mechanism based on file renaming was implemented. To limit infrastructure overload, running tasks are replicated up to 5 times only.

Input file unavailability is distinguished from non-existent file using ad-hoc parsing of standard error files. File replication is implemented differently depending on the incident. In case of input data unavailability, a file is replicated to a storage resource randomly selected in the biomed VO. The maximal allowed number of file replicas is set to 5. In case a site is misconfigured, replication to the site local storage resource is first attempted. This aims at circumventing inter-domain connectivity issues. If there is no local storage available or the replication process fails, then a second attempt is performed to a storage resource successfully accessed by other tasks executed on the same site.

Problematic sites are only temporarily blacklisted during a time interval set

from exponential back-off. The site is first blacklisted for 1 minute only and then put back on the white list. In case it is detected misconfigured again, then the blacklist duration is increased to 2 minutes, then to 4 minutes, 16 minutes, etc.

6.2 Experiment conditions and metrics

Two workflow activities are considered. `FIELD-II/pasa` consists of 122 invocations of an ultrasonic simulator on an echocardiography 2D data set. It is a data-intensive activity where invocations use from a few seconds to some 15 minutes of CPU time; it transfers 208 MB of input data and outputs about 40 KB of data. `Mean-Shift/hs3` has 250 CPU-intensive invocations of an image filtering application. Invocation CPU time ranges from a few minutes up to one hour; input data size is 182 MB and output is less than 1 KB. Files were replicated on two storage sites for both activities.

Two experiments were performed on both workflow activities. Experiment 1 aims at testing that recoverable errors are detected and handled. It is a correct execution where all the input files exist and the application is supposed to run properly and produce the expected results. Five repetitions were performed for each workflow activity.

Experiment 2 aims at testing that unrecoverable errors are quickly identified and the execution is stopped. Unrecoverable errors were intentionally injected in 3 different runs: in run `non-existent inputs`, non-existent file paths were used for all the invocations; in `application-error`, all the file paths existed but input files were corrupted; and in `non-existent output`, input files were correct but the application did not produce the expected results.

MOTEUR was configured to resubmit failed tasks up to 5 times in all runs of both experiments. For each experiment, a workflow execution using our method (`Self-Healing`) was compared to a control execution (`No-Healing`). Executions were launched in production conditions, i.e., without any control of the number of available resources and reliability. `Self-Healing` and `No-Healing` were both launched simultaneously to ensure similar grid conditions. Runs were performed along a time period of one week, therefore under different grid conditions. The DIRAC scheduler was configured to equally distribute resources among executions. We used DIRAC v5r12p9 and MOTEUR 0.9.19.

The waste metric used by Cirne et al. [8] does not fit our context because it cannot provide an effective estimation of the amount of resource wasted by self-healing simulations when compared to the control ones. Here, resource waste is assessed by the amount of resource time consumed by the simulations performing the healing process related to the amount of resource time consumed by control simulations. We use the `waste coefficient` (w), defined as follows:

$$w = \frac{\sum_{i=1}^{n} h_i + \sum_{j=1}^{m} r_j}{\sum_{i=1}^{n} c_i} - 1$$

where h_i and c_i are the resource time consumed (CPU time + data transfers time) by n completed tasks for `Self-Healing` and `No-Healing` simulations respectively, and r_i is the resource time consumed by m unused replicas. Note that task replication usually leads to $h_i \leq c_i$. If $w > 0$, the healing approach is wasting resources compared to the control one. If $w < 0$, the healing approach is

consuming less resources compared to the control one, which can happen when faster resources are selected.

6.3 Results and Discussion

Experiment 1: this experiment was conducted by using both `Slope Contraction` and `Estimation by Median` to detect blocked activities. Figure 9 presents the makespan of `FIELD-II/pasa` and `Mean-Shift/hs3` by using the `Slope Contraction` method to detect blocked activities for the 5 repetitions. The makespan was considerably reduced in all repetitions of both activities. Speed-up values yielded by `Self-Healing` ranged from 2.6 to 4 for `FIELD-II/pasa` and from 1.3 to 2.6 for `Mean-Shift/hs3`.

Figures 10 and 11 present a cumulative density function (CDF) of the number of completed tasks for `FIELD-II/pasa` and `Mean-Shift/hs3`, respectively. In most cases completion curves of both `Self-Healing` and `No-Healing` executions are similar up to 95%. This confirms that both executions had similar grid conditions.

Table 4 shows occurrences of incident levels and associated actions. All recoverable incidents were observed, except $x_{7,2}$. For `FIELD-II/pasa`, $x_{2,2}$ was the predominant incident due to the data-intensive nature of the application. No blocked activity was detected due to important task replication triggered by low efficiency. For `Mean-Shift/hs3`, low efficiency and blocked activity almost equally appeared. The total number of replicated tasks for all repetitions was $1,128$ for `FIELD-II/pasa` (i.e. 1.8 task replication per invocation in average) and 644 for `Mean-Shift/hs3` (i.e. 0.5 task replication per invocation in average).

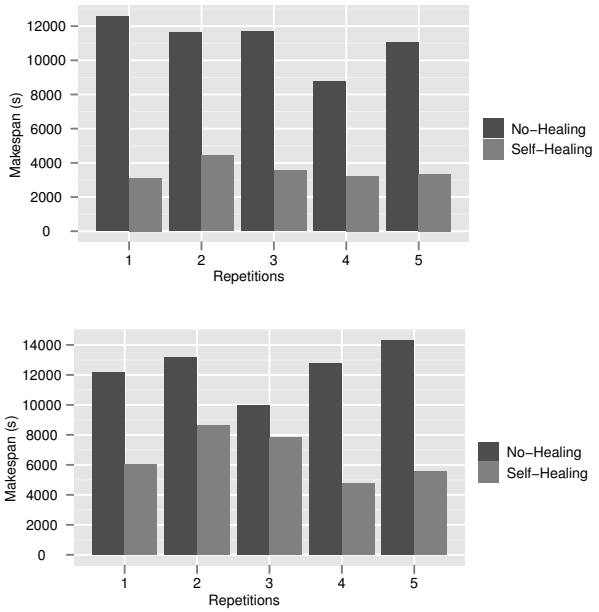


Figure 9: Experiment 1: execution makespan for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) by using the `Slope Contraction` method.

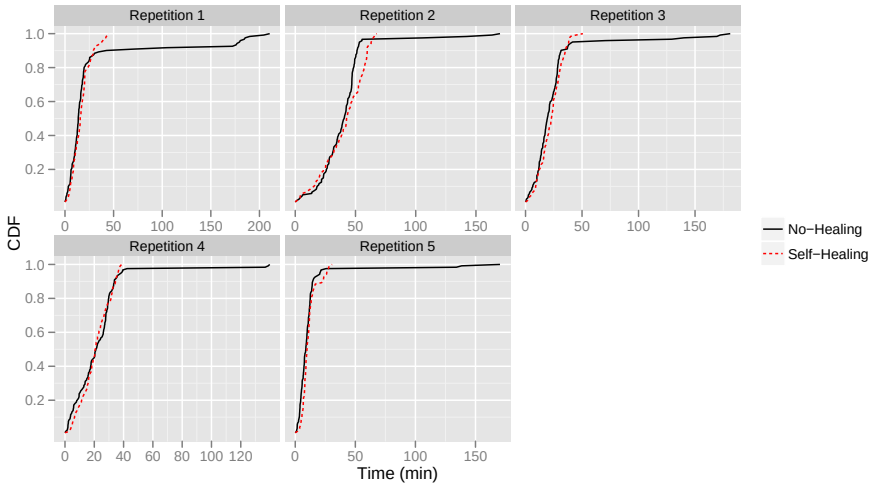


Figure 10: Experiment 1: CDF of the number of completed tasks for `FIELD-II/pasa` repetitions by using the `Slope Contraction` method.

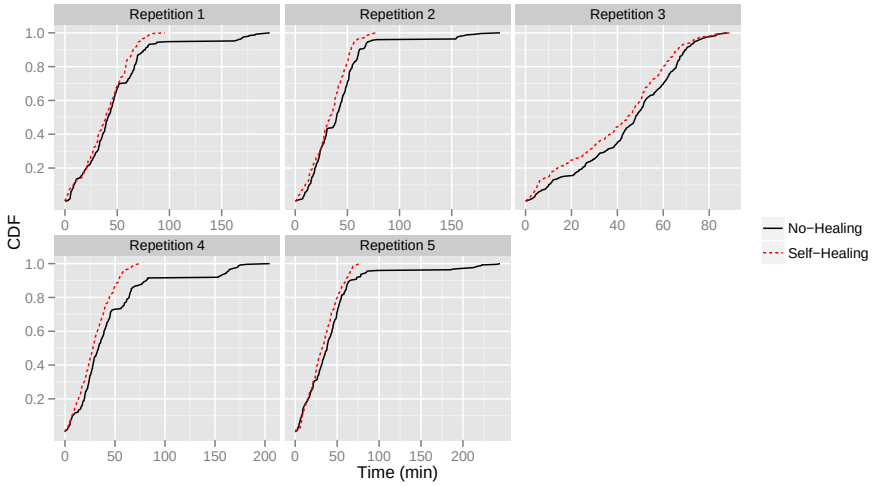


Figure 11: Experiment 1: CDF of the number of completed tasks for `Mean-Shift/hs3` repetitions by using the `Slope Contraction` method.

Tables 5 and 6 show the waste coefficient value for `FIELD-II/pasa` and `Mean-Shift/hs3` respectively. The `Self-Healing` process consumed up to 33% more resources for `FIELD-II/pasa` and 75% for `Mean-Shift/hs3` compared to a control execution. Even if this mechanism properly detects blocked states, it wastes the resources of badly performing tasks that will be overlapped by replicas. This waste of resource is related to (i) a late detection of the blocked state and (ii) a greedy replication process. For instance, the poor performance of a task submitted at the beginning of the execution can be masked by the good performance of the others.

Activity	Incident level	Occurrence	Actions
FIELD-II/pasa	$x_{2,2}$	262	replicate running tasks replicate input files
	$x_{9,2}$	12	blacklist site
Mean-Shift/hs3	$x_{1,2}$	111	replicate running tasks
	$x_{2,2}$	83	replicate running tasks replicate input files
	$x_{5,2}$	16	replicate files on sites
	$x_{5,3}$	6	blacklist site
	$x_{9,2}$	8	blacklist site

Table 4: Experiment 1: occurrences of incident levels (cumulative values for 5 repetitions).

Repetition	h	r	c	w
1	39,035s	28,653s	55,244s	0.22
2	37,035s	5,191s	50,829s	−0.17
3	28,454s	9,594s	28,594s	0.33
4	21,021s	13,764s	27,586s	0.26
5	37,494s	13,438s	42,019s	0.21

Table 5: Waste coefficient values for `FIELD-II/pasa` by using the `Slope Contraction` method.

Figure 12 shows the makespan of `FIELD-II/pasa` and `Mean-Shift/hs3` by using the `Estimation by Median` method to detect blocked activities for the 5 repetitions. The makespan values are comparable to the results presented in Figure 9. This means that blocked tasks are being properly detected and replicated.

Analogously to Figures 10 and 11, Figures 13 and 14 present the CDF of the number of completed tasks for both applications. Again, curves similarity up to 95% indicate similar grid conditions. In some cases (e.g. `Repetition 2` in Figures 10 and 14) `Self-Healing` execution presents lower performance than `No-Healing` execution but it is compensated by the long-tail effect produced by the latter.

Tables 7 and 8 show the waste coefficient values for the 5 repetitions presented in Figure 12 for `FIELD-II/pasa` and `Mean-Shift/hs3` respectively. The `Self-Healing` process reduced resource consumption up to 26% when compared to the control execution. This happens because replication increases the probability to select a faster resource. The total number of replicated tasks for

Repetition	h	r	c	w
1	84,499s	74,917s	95,319s	0.67
2	121,496s	88,963s	129,250s	0.63
3	81,745s	16,418s	88,032s	0.11
4	98,235s	146,016s	141,292s	0.73
5	103,867s	81,614s	105,783s	0.75

Table 6: Waste coefficient values for `Mean-Shift/hs3` by using the `Slope Contraction` method.

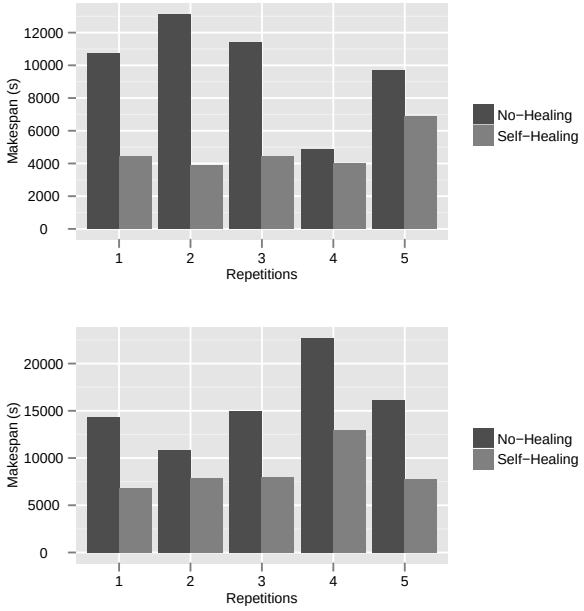


Figure 12: Execution makespan for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) by using the `Estimation by Median` method.

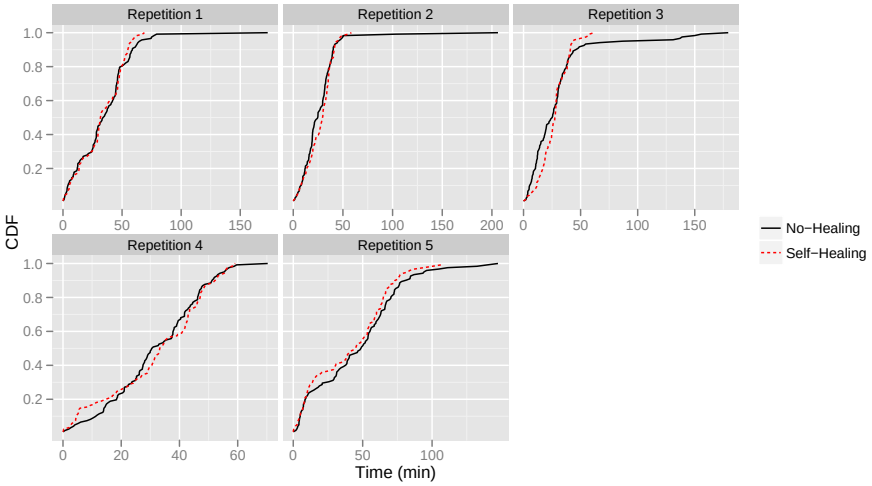


Figure 13: Experiment 1: CDF of the number of completed tasks for `FIELD-II/pasa` repetitions by using the `Estimation by Median` method.

all repetitions was 172 for `FIELD-II/pasa` (i.e. 0.28 task replication per invocation in average) and 308 for `Mean-Shift/hs3` (i.e. 0.24 task replication per invocation in average).

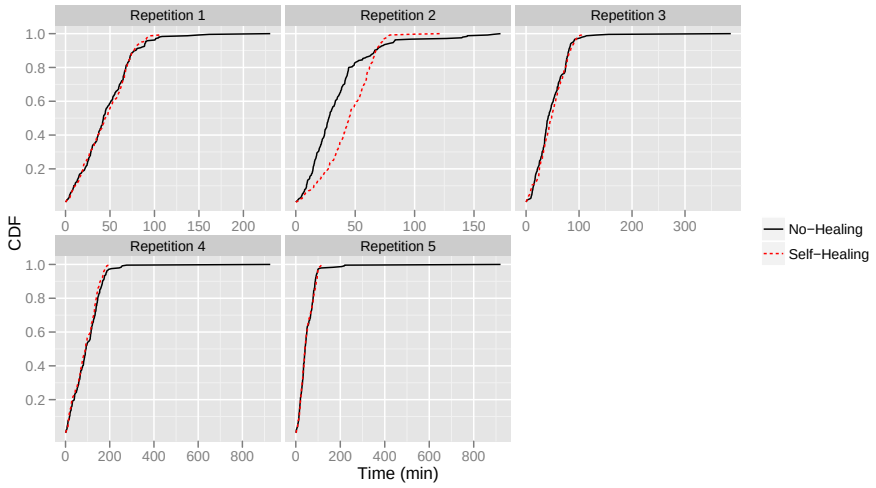


Figure 14: Experiment 1: CDF of the number of completed tasks for `Mean-Shift/hs3` repetitions by using the `Estimation by Median` method.

Repetition	h	r	c	w
1	56,159s	2,203s	64,163s	−0.10
2	60,991s	6,383s	79,031s	−0.15
3	60,473s	10,818s	77,851s	−0.09
4	42,475s	1,420s	41,528s	0.05
5	56,726s	4,527s	82,555s	−0.26

Table 7: Waste coefficient values for `FIELD-II/pasa` by using the `Estimation by Median` method.

Experiment results show that `Slope Contraction` and `Estimation by Median` methods properly detect blocked activities and speed up the execution. The first method consumes more resources when compared to the control execution while the latter reduces resource consumption. Furthermore, by using `Estimation by Median` method no low efficiency incidents were detected. Indeed, it is intrinsically handled by the method since low efficiency causes a time delay.

Repetition	h	r	c	w
1	119,597s	5,778s	126,714s	−0.02
2	125,959s	4,792s	161,493s	−0.20
3	133,935s	14,352s	151,091s	−0.02
4	147,077s	2,898s	152,282s	−0.02
5	141,494s	17,514s	159,152s	−0.01

Table 8: Waste coefficient values for `Mean-Shift/hs3` by using the `Estimation by Median` method.

Experiment 2 Figure 15 shows the makespan of `FIELD-II/pasa` and `Mean-Shift/hs3` for the 3 runs where unrecoverable errors were introduced. `No-Healing` was manually stopped after 7 hours to avoid flooding the infrastructure with faulty tasks. In all cases, `Self-Healing` was able to detect the issue and stop the execution far before `No-Healing`. It confirms that the healing process is indeed able to identify unrecoverable errors and stop the execution accordingly. As shown on Table 9, the number of submitted fault tasks was significantly reduced, which has benefits both to the infrastructure and to the gateway itself.

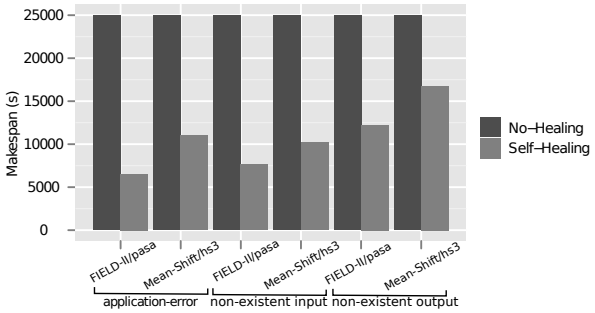


Figure 15: Experiment 2: makespan of `FIELD-II/pasa` and `Mean-Shift/hs3` for 3 different runs.

Run		Number of tasks	
		Self-Healing	No-Healing
application-error	FIELD-II/pasa	196	732
	Mean-Shift/hs3	249	1500
non-existent input	FIELD-II/pasa	293	732
	Mean-Shift/hs3	417	1500
non-existent output	FIELD-II/pasa	287	732
	Mean-Shift/hs3	364	1500

Table 9: Number of submitted faulty tasks.

7 Conclusion

We presented a simple, yet practical method for autonomous detection and handling of operational incidents in workflow activities. No strong assumption is made on the task duration or resource characteristics and incident degrees are measured with metrics that can be computed online. We made the hypothesis that incident degrees were quantified into distinct levels, which we verified using extensive historical information. Incident levels are associated (offline) to action sets ranging from light execution tuning (file/task replication) to radical site blacklisting or activity interruption. Action sets are selected based on the degree of their associated incident level and on confidence of association rules determined from execution history.

This strategy was implemented in the MOTEUR workflow engine and deployed on the European Grid Infrastructure with the DIRAC resource manager.

Results show that the proposed method speeds up execution up to a factor of 4, consumes up to 26% less resource time than a control execution and properly detects unrecoverable errors.

The approach can be extended in several ways. First, other incidents could be added, provided that they can be quantified online by a metric ranging from 0 to 1. Possible candidates are infrastructure service downtimes (e.g. file catalog, storage servers, computing sites) detected by external active monitoring systems such as Nagios [21]. Action sets could also be extended, for instance with actions related to resource provisioning.

Besides, mode detection used for incident quantification could be improved by (i) automated detection (e.g. with Mean-Shift [10]) and (ii) periodical update from execution history. Using the history of actions performed to adjust incident degree could also be envisaged. For instance, incidents for which several actions already have been taken could be considered more critical.

Finally, other components of science-gateways could be targeted with the same approach. Our future work addresses complete workflow executions, taking actions such as pausing workflow executions, detected blocked workflows beyond activities, or allocating resources to users and executions.

8 Acknowledgment

This work is funded by the French National Agency for Research under grant ANR-09-COSI-03 "VIP'. We thank the European Grid Initiative and National Grid Initiatives, in particular France-Grilles, for providing the infrastructure and technical support. We also thank Ting Li and Olivier Bernard for providing optimization use-cases to the Virtual Imaging Platform.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD '93 Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.

[2] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger, and F. Zini. Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 661, 2003.

[3] A. Benoit, L. Marchal, J.-F. Pineau, Y. Robert, and F. Vivien. Scheduling Concurrent Bag-of-Tasks Applications on Heterogeneous Platforms. *IEEE Transactions on Computers*, 59:202–217, 2010.

[4] L. Broto, D. Hagimont, P. Stolf, N. De Palma, and S. Temate. Autonomic Management Policy Specification in Tune. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1658–1663, New York, NY, USA, 2008.

[5] S. Camarasu-Pop, T. Glatard, J. T. Moscicki, H. Benoit-Cattin, and D. Sarrut. Dynamic Partitioning of GATE Monte-Carlo Simulations on EGEE. *Journal of Grid Computing*, 8(2):241–259, mar 2010.

[6] H. Casanova. On the Harmfulness of Redundant Batch Requests. *International Symposium on High-Performance Distributed Computing*, 0:255–266, 2006.

[7] H. Casanova, M. Gallet, and F. Vivien. Non-clairvoyant Scheduling of Multiple Bag-of-Tasks Applications. In *Euro-Par 2010 - Parallel Processing*, volume 6271 of *Lecture Notes in Computer Science*, pages 168–179, 2010.

[8] W Cirne, F. Brasileiro, D. Paranhos, L.F.W. Goes, and W. Voorsluys. On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems. *Parallel Computing*, 33:213–234, 2007.

[9] P. Collet, F. Křikava, J. Montagnat, M. Blay-Fornarino, and D. Manset. Issues and Scenarios for Self-Managing Grid Middleware. In *Workshop on Grids Meet Autonomic Computing, in association with ICAC'2010*, Washington, DC, USA, June 2010. ACM.

[10] D. Comaniciu and P. Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[11] K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975. AAI7609381.

[12] A.H. Elghirani, R. Subrata, and A.Y. Zomaya. A Proactive Non-Cooperative Game-Theoretic Framework for Data Replication in Data Grids. In *8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, page 433, 2008.

[13] T. Elteto, C. Germain-Renaud, P. Bondon, and M. Sebag. Towards Non-Stationary Grid Models. *Journal of Grid Computing*, December 2011.

[14] R. Ferreira da Silva, S. Camarasu-Pop, Baptiste Grenier, Vanessa Hamar, David Manset, Johan Montagnat, Jérôme Revillard, Javier Rojas Balderrama, Andrei Tsaregorodtsev, and T. Glatard. Multi-Infrastructure Workflow Execution for Medical Simulation in the Virtual Imaging Platform. In *HealthGrid 2011*, Bristol, UK, 2011.

[15] R. Ferreira da Silva and T. Glatard. A Science-Gateway Workload Archive to Study Pilot Jobs, User Activity, Bag of Tasks, Task Sub-Steps, and Workflow Executions. In *CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing*, Rhodes, GR, 2012.

[16] R. Garg and A.K. Singh. Fault Tolerance in Grid Computing: State of the Art and Open Issues. *International Journal of Computer Science & Engineering Survey (IJCSES)*, 2(1), 2011.

[17] C. Germain-Renaud, A. Cady, P. Gauron, M. Jouvin, C. Loomis, J. Martyniak, J. Nauroy, G. Philippon, and M. Sebag. The Grid Observatory. *IEEE International Symposium on Cluster Computing and the Grid*, pages 114–123, 2011.

[18] S. Gesing and J. van Hemert, editors. *Concurrency and Computation: Practice and Experience, Special Issue on International Workshop on Portals for Life-Sciences 2009*, volume 23:3, march 2011.

[19] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and Efficient Workflow Deployment of Data-Intensive Applications on Grids with MOTEUR. *International Journal of High Performance Computing Applications (IJHPCA)*, 22(3):347–360, August 2008.

[20] C.-C. Hsu, K.-C. Huang, and F.-J. Wang. Online Scheduling of Workflow Applications in Grid Environments. *Future Generation Computer Systems*, 27(6):860 – 870, 2011.

[21] E. Imamagic and D. Dobrenic. Grid Infrastructure Monitoring System Based on Nagios. In *Proceedings of the 2007 workshop on Grid monitoring*, pages 23–28, New York, NY, USA, 2007.

[22] A. Iosup and D. Epema. Grid Computing Workloads. *Internet Computing, IEEE*, 15(2):19 –26, march-april 2011.

[23] P. Kacsuk. P-GRADE Portal Family for Grid Infrastructures. *Concurrency and Computation: Practice and Experience*, 23(3):235–245, 2011.

[24] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41 – 50, jan 2003.

[25] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 398 –407, may 2010.

[26] D. Lingrand, J. Montagnat, J. Martyniak, and D. Colling. Analyzing the EGEE production grid workload: application to jobs submission optimization. In *14th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'09)*, pages 37–58, Roma, Italy, May 2009.

[27] D.S. Malik, John N. Mordeson, and M.K. Sen. On Subsystems of a Fuzzy Finite State Machine. *Fuzzy Sets and Systems*, 68(1):83 – 92, 1994.

[28] H. Nguyen Van, F. Dang Tran, and J.-M. Menau. Autonomic Virtual Resource Management for Service Hosting Platforms. In *Workshop on Software Engineering Challenges in Cloud Computing*, 2009.

[29] J.-N. Quintin and F. Wagner. WSCOM: Online Task Scheduling with Data Transfers. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:344–351, 2012.

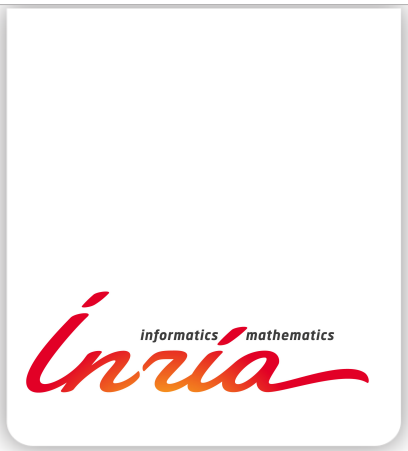
[30] J. Rehn, T. Barrass, D. Bonacorsi, J. Hernandez, I. Semeniouk, L. Tuura, and Y. Wu. PhEDEx High-Throughput Data Transfer Management System. In *Computing in High Energy Physics, CHEP'2006*, 2006.

[31] S. Shahand, M. Santcroos, Y. Mohammed, Vladimir Korkhov, Angela C.M Luyf, Antoine van Kampen, and Sílvia D. Olabarriaga. Front-ends to Biomedical Data Analysis on Grids. In *Proceedings of HealthGrid 2011*, Bristol, UK, june 2011.

[32] E. Stehle, K. Lynch, M. Shevertalov, C. Rorres, and S. Mancoridis. On the Use of Computational Geometry to Detect Software Faults at Runtime. In *Proceeding of the 7th international conference on Autonomic computing*, pages 109–118, New York, NY, USA, 2010.

[33] A. Tsaregorodtsev, N. Brook, A. Casajus Ramo, P. Charpentier, J. Closier, G. Cowan, R. Graciani Diaz, E. Lanciotti, Z. Mathe, R. Nandakumar, S. Paterson, V. Romanovsky, R. Santinelli, M. Sapunov, A.C. Smith, M. Seco Miguelez, and A. Zhelezov. DIRAC3. The New Generation of the LHCb Grid Software. *Journal of Physics: Conference Series*, 219(6):062029, 2009.

[34] X. Zhang, C. Germain-Renaud, and M. Sebag. Adaptively Detecting Changes in Autonomic Grid Computing. In *Procs of ACS 2010*, Belgique, October 2010.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399